Appendix for "SE(2)-Equivariant Pushing Dynamics Models for Tabletop Object Manipulations"

Seungyeon Kim[1] **Byeongdo Lim**[1] **Yonghyeon Lee**[1] **Frank Chongwoo Park**[1,2]
[1]Seoul National University, [2]Saige Research
{ksy, bdlim, yhlee}@robotics.snu.ac.kr, fcp@snu.ac.kr

A Related Works

A.1 Model-free Pushing Manipulation

In model-free pushing manipulation methods, a policy that directly maps a visual observation to a sequence of pushing actions is learned without learning a pushing dynamics model. They typically require to design a task-specific reward function and train the policy in an end-to-end manner with a large amount of data. Researchers have attempted to solve diverse tasks with these methods such as grasping [1, 2, 3], singulation [4], object rearrangement [5, 6], object sorting [7, 8], and invisible target object finding [9, 10]. Model-free methods are known to have very good convergence performance, but they require a lot of data. Also, when a new task is given, the agent must be trained from scratch. On the other hand, in model-based approaches, the learned model is reusable given a new task, and much less additional data is needed.

A.2 Visual Pushing Dynamics Learning

A few studies have proposed data-driven *visual* pushing dynamics models using trained deep neural network models. Early works have modeled the object motions with several rigid body transformations instead of predicting pixel-wise flow from observation [11, 12]. Some works have proposed object-centric models by representing each object by a visual feature [13, 14, 15], or a segmented image [16]. However, these works predict the motions of the objects from only partial observation, so they often predict inaccurate or incomplete object motions. A recent work infers the amodal 3D geometry of each object on 3D voxel space and uses it to predict the motions to increase the accuracy and solve the incompleteness of the above methods [17]. Our work is also in the spirit of [17] in utilizing the ground-truth information of the objects to increase the accuracy of the prediction. Still, we use implicit representation for the objects to utilize them for efficient motion prediction and various manipulations.

A.3 Shape Recognition from Visual Observation

Many works have been proposed to recognize the full 3D shapes from partial observations such as depth images. Some of them use explicit representation such as occupancy grid [18], point cloud [19], or mesh [20]. Since they often lead to shape prediction not being precise enough due to limited resolutions of the representations, recent works have explored learning implicit 3D representations for the objects using neural implicit functions [21, 22, 23, 24]. In this paper, we adopt superquadric functions that balance the expressiveness of the shapes and efficiency of the computational with a small number of parameters [25]. They have been used for robotic manipulation such as grasping [26, 27, 28]. We note in our paper that we represent each object as a single superquadric function, but our work can be easily extended to general implicit representations, especially deformable superquadric [29, 30] or a set of superquadrics [31].

A.4 Invariance and Equivariance in Deep Learning and Robot Manipulations

Recently, invariance and equivariance properties turn out to be very important inductive biases for deep learning models to generalize well and be trained data efficiently [32]. Convolutional neu-

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

ral networks (CNNs) have translation equivariance properties, which are particularly suitable for image recognition tasks [33]. Graph neural networks, point cloud neural networks, and set neural networks have the permutation invariant properties [34, 35, 36]. More advanced equivariance properties such as rotational equivariance for image data and $\mathrm{SO}(3)$-equivariance for spherical image data have been achieved by group equivariant CNNs [37, 38, 39]. In the context of robot manipulations, recent works have adopted these equivariance principles and shown significantly improved sample efficiency and performance. Some of them use group invariant Q function to achieve the group equivariant reinforcement learning [40**?** , 41] or use $\mathrm{SE}(3)$-equivariant object representation for object manipulation tasks [42]. Our work differs from other works in that we learn *equivariant dynamics model* for robot pushing manipulation, and it is novel in that the $\mathrm{SE}(2)$-equivariant dynamics model, which is a suitable inductive bias for pushing dynamics, is formulated. Similar to our approach, there is an approach that performs input transformations to achieve the equivariance [43], but the model is equivariant for only a few rotations and scales; our model is intrinsically equivariant to the continuous SE(2) transformations.

B Details for SE(2)-Equivariant Pushing Dynamics Model

B.1 Pose Decomposition

In the manuscript, we introduce an object pose decomposition method that decomposes an object pose $\mathbf{T} \in \mathrm{SE}(3)$ to a projected pose to the table denoted by $\mathbf{C} \in \mathrm{SE}(3)$ and the remaining rigid-body transformation $\mathbf{U} \in \mathrm{SE}(3)$ such that $\mathbf{T} = \mathbf{CU}$ as shown in the left of Figure 1.

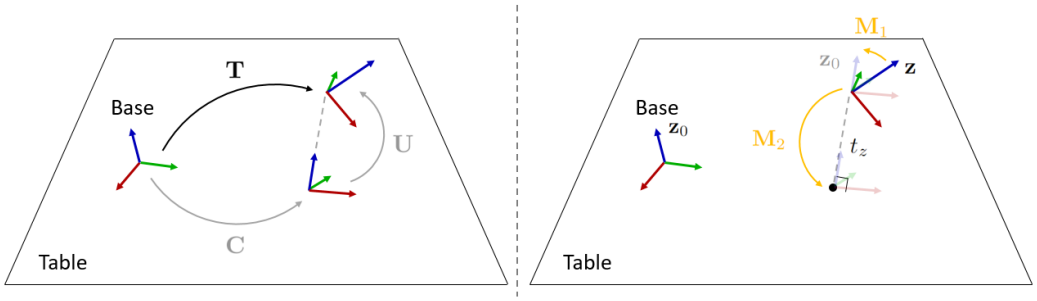

Figure 1: Object Pose Decomposition.

To achieve this, we first calculate the projection matrix $\mathbf{U}^{-1}$ as shown in the right of Figure 1. The projection matrix is decomposed as $\mathbf{M}_1\mathbf{M}_2$, where $\mathbf{M}_1 \in \mathrm{SE}(3)$ is the rotation matrix that aligns the z-axis of the object pose with the z-axis of the base frame and $\mathbf{M}_2 \in \mathrm{SE}(3)$ is the translation matrix that projects the z-axis-aligned frame to the table surface. If $\mathbf{M}_1\mathbf{M}_2$ is calculated, we can obtain $\mathbf{U} = (\mathbf{M}_1\mathbf{M}_2)^{-1}$ and $\mathbf{C} = \mathbf{T}\mathbf{M}_1\mathbf{M}_2$ accordingly.

To calculate $\mathbf{M}_1$, we follow the following steps. The vector $\mathbf{z}$ and $\mathbf{z_0}$ is the z-axis vector of the base frame $\mathbf{T}_0$ and object frame $\mathbf{T}$ expressed in the base frame $\mathbf{T}_0$ (e.g., $\mathbf{z_0} = (0, 0, 1)^T$). First, we calculate the inner product and cross product between $\mathbf{z}$ and $\mathbf{z_0}$ to obtain the following results:

$$\cos\phi = \mathbf{z} \cdot \mathbf{z}_0, \quad \sin\phi = ||\mathbf{z} \times \mathbf{z}_0||, \quad \mathbf{w} = \frac{\mathbf{z} \times \mathbf{z}_0}{||\mathbf{z} \times \mathbf{z}_0||},$$

where $\mathbf{w}$ is the rotation axis expressed in $\mathbf{T}_0$ and ϕ is the rotation angle; ϕ can be obtained by $\phi = \mathrm{atan2}(\sin\phi, \cos\phi)$. Then the matrix $\mathbf{M}_1$ is of the form

$$\mathbf{M}_1 = \begin{bmatrix} \exp([\mathbf{R}^{-1}\mathbf{w}]\phi) & 0 \\ 0 & 1 \end{bmatrix}, \tag{1}$$

where $\mathbf{R} \in \mathrm{SO}(3)$ is the rotation matrix part of $\mathbf{T}$, the bracket $[\cdot] : \mathbb{R}^3 \to \mathrm{so}(3)$ is the skew-symmetric operation, and $\exp : \mathrm{so}(3) \to \mathrm{SO}(3)$ is the exponential map from rotation vector $\mathrm{so}(3)$ to rotation matrix $\mathrm{SO}(3)$. In other words, $\mathbf{M}_1$ rotates the frame $\mathbf{T}$ by ϕ with the rotation axis $\mathbf{R}^{-1}\mathbf{w}$ which is the axis of the rotation expressed in $\mathbf{T}$. The matrix $\mathbf{M}_2$ is simply of the form

$$\mathbf{M}_2 = \begin{bmatrix} \mathbf{I}_3 & \mathbf{t_z} \\ 0 & 1 \end{bmatrix}, \tag{2}$$

where $\mathbf{t_z} = (0, 0, -t_z) \in \mathbb{R}^3$. We note that (i) $\mathbf{M}_1$ and $\mathbf{M}_2$ are uniquely defined for given $\mathbf{T}$, (ii) the projected transformation matrix $\mathbf{C} = \mathbf{T}\mathbf{M}_1\mathbf{M}_2$ is of the form

$$\mathbf{C} = \begin{bmatrix} \mathbf{Rot}(\hat{\mathbf{z}}, \theta) & \mathbf{t_{xy}} \\ 0 & 1 \end{bmatrix}, \tag{3}$$

where $\mathbf{Rot}(\hat{\mathbf{z}}, \theta) \in \mathrm{SO}(3)$ is a 3×3 rotation matrix for rotations around z-axis and $\mathbf{t_{xy}} = (t_x, t_y, 0) \in \mathbb{R}^3$, and (iii) given a new object frame $\mathbf{T}' = \mathbf{C}_a\mathbf{T}$ ($\mathbf{C}_a$ has the form of Equation (3)) and $\mathbf{T} = \mathbf{CU}$, the frame is uniquely expressed by $\mathbf{T}' = \mathbf{C}'\mathbf{U}$ where $\mathbf{C}' = \mathbf{C}_a\mathbf{C}$.

B.2 Proof for Equivariance

Proposition 1. *A pushing dynamics model $f = \{f_i\}_{i=1}^N$ proposed in Section 2 is* SE(2)*-equivariant, i.e., given the inputs and outputs*

$$\{\mathbf{T}'_i\}_{i=1}^N = f(\{(\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{p}, \mathbf{v})), \tag{4}$$

and for all rigid-body transformations that have the following form

$$\mathbf{C} = \begin{bmatrix} \mathbf{Rot}(\hat{\mathbf{z}}, \theta) & \mathbf{t_{xy}} \\ 0 & 1 \end{bmatrix}, \tag{5}$$

where $\mathbf{Rot}(\hat{\mathbf{z}}, \theta)$ is a 3×3 rotation matrix for rotations around z-axis and $\mathbf{t_{xy}} = (t_x, t_y, 0) \in \mathbb{R}^3$, the model satisfies

$$\{\mathbf{CT}'_i\}_{i=1}^N = f(\{(\mathbf{CT}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t_{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v})). \tag{6}$$

Proof. It is enough to show that one element f_i is SE(2)-equivariant. For convenience, Figure 3 describing f_i from the manuscript is excerpted below.

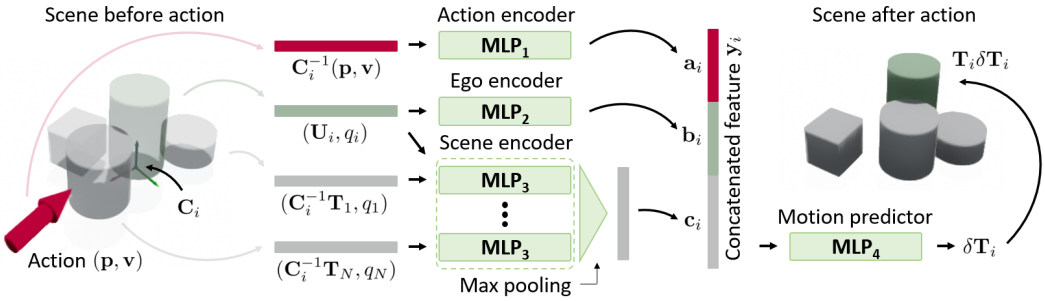

Figure 2: SE(2)-equivariant pushing dynamics neural network architecture for an i-th object, f_i.

Since $\mathbf{CT}'_i = (\mathbf{CT}_i)\delta\mathbf{T}_i$, the claim that "$f_i$ is SE(2)-equivariant" is equivalent to the claim that "$\delta\mathbf{T}_i$ is invariant to the arbitrary transformation $\mathbf{C}$". This claim suffices to show that the network inputs are invariant to the transformation $\mathbf{C}$. Below is a description of whether each input is invariant. We note that when $\mathbf{T}_i$ is decomposed to $\mathbf{T}_i = \mathbf{C}_i\mathbf{U}_i$, the frame $\mathbf{CT}_i$ is also decomposed to $\mathbf{CT}_i = (\mathbf{CC}_i)\mathbf{U}_i$ from the property (iii) above.

Action. Through a series of calculations, we below confirm that the term for the action is invariant.

$$\begin{aligned} &(\mathbf{CC}_i)^{-1}(\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t_{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v}) \\ &= \mathbf{C}_i^{-1}\mathbf{C}^{-1}(\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t_{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v}) \\ &= ... \\ &= \mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v}) \end{aligned}$$

Ego. The frames $\mathbf{U}_i$ are invariant.

Scene. Since $(\mathbf{CC}_i)^{-1}(\mathbf{CT}_j) = \mathbf{C}_i^{-1}\mathbf{T}_j$ for $j = 1, ..., N, j \neq i$, the terms for the surrounding objects are also invariant.

In conclusion, $\delta\mathbf{T}_i$ is invariant to the arbitrary transformation $\mathbf{C}$, so f_i is SE(2)-equivariant. Therefore, the pushing dynamics model $f = \{f_i\}_{i=1}^N$ described in Section 2 is SE(2)-equivariant. □

C Implementation Details for Our Overall Methods

C.1 Object Shape and Pose Recognition

The goal of the object shape and pose recognition is to design an algorithm that takes a partial point cloud of the objects in the scene $\mathcal{P} \subset \mathbb{R}^3$, observed from a (synthetic or real-world) depth camera, as input and outputs the superquadric representations $\{\mathbf{q}_i, \mathbf{T}_i\}_{i=1}^N$, where $\mathbf{q}_i$ is the shape parameter, $\mathbf{T}_i \in \mathrm{SE}(3)$ is the pose, and N is the number of the objects. We note that a noise is added to each point $\mathbf{x} \in \mathcal{P}$ – in detail, $\mathbf{x} \mapsto \mathbf{x} + m\mathbf{v}$ where $\mathbf{v}$ is uniformly sampled on $\mathbb{S}^2$ and m is sampled from a Gaussian with zero-mean and standard deviation 0.001 – to bridge the sim-to-real gap on vision sensor data as described in [44, 45]. To achieve this goal to design the algorithm, we first segment a partially observed point cloud $\mathcal{P}$ into a set of object point clouds $\{\mathcal{P}_i\}_{i=1}^N$, then convert them into superquadric representations $\mathbf{q}_i$ and $\mathbf{T}_i$.

From the raw vision sensor data $\mathcal{P} \subset \mathbb{R}^3$, the table points are discarded through plane fitting and then up/down-sampled to 2048 points. In the other words, the partially observed point cloud $\mathcal{P}$ is processed to $\mathcal{P} = \{\mathbf{x}_j \in \mathbb{R}^3\}_{j=1}^n$, where $n = 2048$.

Point cloud segmentation. We use the same architecture and loss function used in [46]. Since our purpose is just to separate the point cloud, the network should learn permutation-invariant segmentation labels, so the loss function should be invariant by a permutation of prediction. To achieve this, we first find a bipartite matching between ground-truth and prediction segmentation labels using the Hungarian algorithm [47], then compute the usual segmentation loss between the matched labels. This makes the loss function permutation-invariant, and guarantees that the network can separate the point cloud properly. The trained segmentation network separates the point cloud $\mathcal{P}$ into several object point clouds $\{\mathcal{P}_i\}_{i=1}^N$.

Superquadric recognition. Our remaining goal is to convert each segmented point cloud $\mathcal{P}_i$ to superquadric representation $\mathbf{q}_i$ and $\mathbf{T}_i$. We first construct the input representation using not only the segmented point cloud $\mathcal{P}_i$ but also surrounding point cloud $\mathcal{P}_1, \cdots, \mathcal{P}_{i-1}, \mathcal{P}_{i+1}, \cdots, \mathcal{P}_N$. In detail, we concatenate 1 after each segmented point $\mathbf{x}_j \in \mathcal{P}_i$ (i.e., $\mathbf{x}_j = (x, y, z) \mapsto (x, y, z, 1)$), and 0 after each surrounding point $\mathbf{x}_j \in \mathcal{P} \backslash \mathcal{P}_i$. We denote this newly created 4-dimensional point from a point $\mathbf{x}_j \in \mathbb{R}^3$ as $\mathbf{x}_{s,j} \in \mathbb{R}^4$, and denote the set of all these points as $\mathcal{P}_{s,i} = \{\mathbf{x}_{s,j} \in \mathbb{R}^4\}_{j=1}^n$; the set $\mathcal{P}_{s,i}$ still has n points.

Then, inspired from [30], we design a neural network that takes the point cloud $\mathcal{P}_{s,i} = \{\mathbf{x}_{s,j} \in \mathbf{R}^4\}_{j=1}^n$ as input and outputs the superquadric parameter $\mathbf{q}_i$ and its pose $\mathbf{T}_i$ that best represents the full object shape as shown in Figure 3. The network consists of (i) the EdgeConv layers [46] with latent space dimension (64, 64, 128, 256) and max pooling operator to produce a global feature vector from $\mathcal{P}_{s,i}$ in a permutation-invariant manner and (ii) four fully-connected layers (MLP) with latent space dimension (512, 256) (with LeakyRelu nonlinearities) to obtain the superquadric parameter $\{a_1, a_2, a_3, e_1, e_2\}$ and the pose $\mathbf{T} = [\mathbf{R}; \mathbf{t}]$ from the extracted global feature. Especially, each MLP outputs (i) translation vector $\mathbf{t} \in \mathbb{R}^3$, (ii) quaternion vector $\mathbf{r} \in \mathbb{S}^3$ representing the rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, (iii) size parameters $\mathbf{a} = (a_1, a_2, a_3) \in \mathbb{R}^3$, and (iv) shape parameters $\mathbf{e} = (e_1, e_2) \in \mathbb{R}^2$; the values e_1 and e_2 are bounded in $[0.2, 1.7]$ since the superquadric equation diverges when e_1 and e_2 goes to zero and shows too complex shapes when e_1 and e_2 become large.

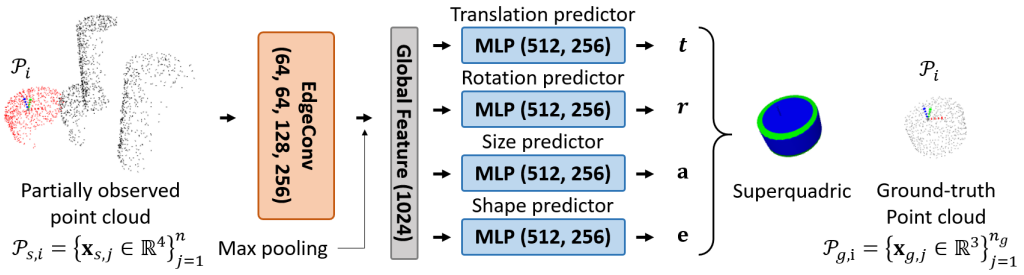

Figure 3: Superquadric Recognition network. The red dots are the points with label 1 and the black dots are the points with label 0 in the partially observed point cloud.

For the predicted superquadric to fit well with the ground-truth shape, the loss function should also be designed to be the difference between the prediction and the ground-truth object shapes. For ground-truth shape, we uniformly sample the points from the surface of the object by $\mathcal{P}_{g,i} = \{\mathbf{x}_{g,j} \in \mathbb{R}^3\}_{j=1}^{n_g}$, where $n_g = 512$, and we call $\mathcal{P}_{g,i}$ the ground-truth point cloud of the i'the object. Then we use the distances from the ground-truth point cloud to the predicted superquadric as the loss function. The distance form is from [48] which is defined as follows. Only in this Appendix C.1, the notation for S is abused as follows:

$$S(x, y, z) = \left(\left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}}, \tag{7}$$

Then, the distance δ between a point $\mathbf{x}_0 \in \mathbb{R}^3$ and a superquadric surface $S(\mathbf{x}) - 1 = 0$ is

$$\delta(\mathbf{x}_0, S) = ||\mathbf{x}_0|| \left| 1 - S^{-\frac{e_1}{2}}(\mathbf{x}_0) \right|, \tag{8}$$

where $\| \cdot \|$ denotes the Euclidean norm. Accordingly, the loss function is defined as:

$$\mathcal{L} = \frac{1}{n_g} \sum_{j=1}^{n_g} \delta^2(\mathbf{T}^{-1}\mathbf{x}_{g,j}, S), \tag{9}$$

where S is defined by the superquadric parameters $\{a_1, a_2, a_3, e_1, e_2\}$ and $\mathbf{T}$ is its pose.

C.2 Details for SuperQuadric Pushing Dynamics Network (SQPD-Net)

SuperQuadric Pushing Dynamics Network (SQPD-Net) has the structure of SE(2)-equivariant pushing dynamics model described in Section 2. The detail of the network architecture is shown in Figure 4. The input dimensions are as follows: (i) the planar pushing action $\mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v})$ is represented by a 5-dimensional vector where the start point is $\mathbf{p} \in \mathbb{R}^3$ and the direction $\mathbf{v}$ is represented by $(\cos\theta, \sin\theta) \in \mathbb{R}^2$ where θ is the pushing direction in the x-y plane, (ii) the i'the object $(\mathbf{U}_i, \mathbf{q}_i)$ is a 12-dimensional vector where $\mathbf{U}_i \in \mathrm{SE}(3)$ is expressed by 7-dimensional (3-dimensional translation and 4-dimensional rotation quaternion) and $\mathbf{q}_i \in \mathbb{R}^5$, and (iii) the surrounding objects $\mathbf{C}_i^{-1}\mathbf{T}_j$ for $j = 1, ..., N, j \neq i$ is also 12-dimensional similar to (ii).

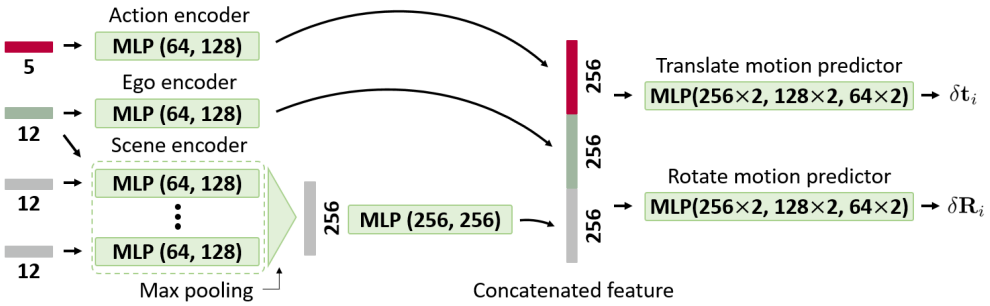

Figure 4: Detail network architecture of SQPD-Net.

Action encoder, ego encoder, and scene encoder consist of the shallow MLP layers with latent space dimension (64, 128) and output feature dimension 256; for scene encoder, an additional MLP layer with latent dimension (256, 256) is also included. After the three feature vectors are obtained from the inputs, they are concatenated and a 768-dimensional global feature is obtained. The global feature then passes through the last MLP layers with size (256, 256, 128, 128, 64, 64) to produce the motion $\delta\mathbf{T}_i = [\delta\mathbf{R}_i; \delta\mathbf{t}_i]$, consists of the translate motion vector $\delta\mathbf{t}_i$ and rotation motion vector $\delta\mathbf{R}_i$ expressed in quaternion. In this work, we consider the planar pushing motions of the objects, so the predicted motion $\delta\mathbf{T}_i$ is of the form Equation (3). We note that all MLPs are followed by LeakyRelu nonlinearities.

Distance measure on $\mathrm{SO}(3)$**.** The general distance measure is the Frobenius norm of the difference of two rotation matrices as follows:

$$d_{\mathrm{SO}(3)}(\mathbf{R}_1, \mathbf{R}_2) = \|\mathbf{R}_1 - \mathbf{R}_2\|_F, \tag{10}$$

where $\mathbf{R}_1, \mathbf{R}_2 \in \mathrm{SO}(3)$ are the rotation matrices. We can use this distance measure for training, but we only consider planar pushing motions of the objects in this paper, we use simpler distance metric as follows:

$$d_{\mathrm{SO}(3)}(\mathbf{R}_1, \mathbf{R}_2) = 1 - \cos(\theta_1 - \theta_2), \tag{11}$$

where θ_1 and θ_2 are the rotation angle of $\mathbf{R}_1$ and $\mathbf{R}_2$, when the rotation matrices are represented as $\mathbf{R}_i = \mathbf{Rot}(\hat{\mathbf{z}}, \theta_i)$

Shape alignment. Since we use symmetric shapes for the object sets, various solutions can appear with the different poses of objects when recognition is performed. We introduce a processing technique that standardizes the recognized object shapes in terms of their poses to reduce the complexity of data statistics and accelerate the training of the SQPD-Net.

We recall the superquadric equation for convenience of explanation:

$$S(x, y, z; \mathbf{q}, \mathbf{I}_4) = \left(\left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}} - 1 = 0. \tag{12}$$

If shape parameters e_1 and e_2 of the predicted superquadric object are almost equal (i.e., $||e_1 - e_2|| < 0.01$), we rearrange the object pose so that the z-axis is close to the surface normal vector of the table and x-axis is close to the action vector. As the object poses are rearranged, the size parameters a_1, a_2, a_3 are also rearranged, e.g., if the x-, y-, and z-axis of the original object pose are rearranged to z-, x-, and y-, the size parameters are changed by $(a_1, a_2, a_3) \mapsto (a_3, a_1, a_2)$. Otherwise, there is no solution for z-axis alignment. So we only rearrange the x-axis of the object pose to be close to the action vector.

C.3 Details for Grasping Cost Function

This subsection includes the details for the calculation of the grasping cost function.

Candidate grasp poses. When a superquadric representation of the target object is obtained from the shape and pose recognition, we generate candidate grasp poses. Grasp poses can be generated in a general superquadric through sampling-based methods [30]; in this work, we use a simple rule-based method for grasp pose generation. We generate top-down and side grasp poses (with 4 directions) according to the shape of the superquadric. At this time, the two gripper fingers should be on the antipodal points on the object. For each approaching direction, 6 grasp poses are generated (maximum 30 grasp poses). Grasp poses with a distance between the antipodal points greater than 7cm are removed from the candidates (the maximum gripper width of the Franka gripper is 8cm). The grasp poses generated from various object shapes are shown in Figure 5.

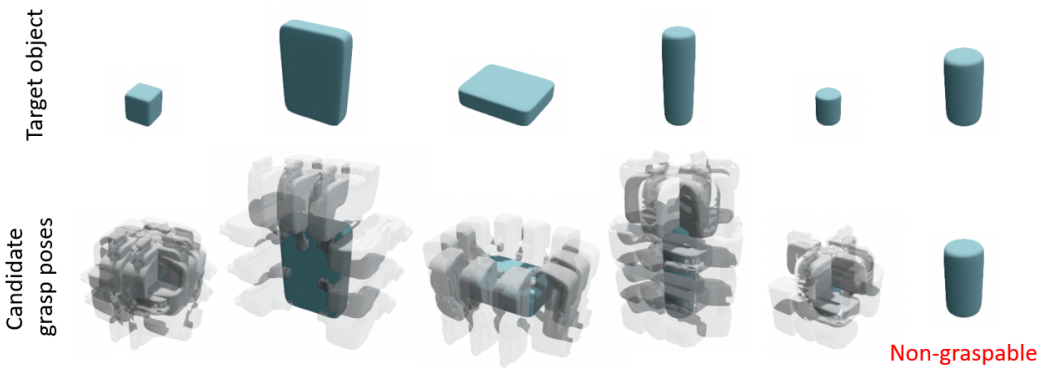

Figure 5: Generated candidate grasp poses for various recognized superquadric shapes.

Gripper collision detection. After generating candidate grasp poses, we check the collisions with the environment and the other recognized objects. The collision detection has to be computed thousands of times to calculate the cost in one step of sampling-based MPC (exactly, it is the product of the number of sampled actions and the number of grasp poses generated above), so it is difficult to use the traditional collision checking algorithm between meshes. Instead, we introduce an efficient method that utilizes the advantage of shape recognition through an implicit function.

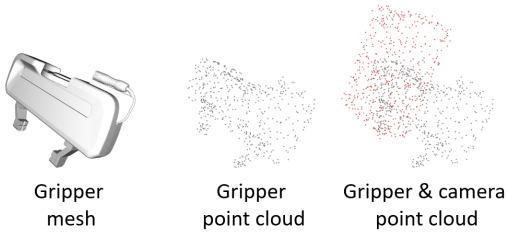

Figure 6: Gripper mesh, sampled gripper point cloud from the mesh, and point cloud with the camera's point cloud.

We first sample the points on the gripper mesh in the maximum open state as shown in Figure 6; the sampled points are denoted by $\mathcal{P}_{gr} = \{\mathbf{x}_{gr,j} \in \mathbf{R}^3\}_{j=1}^{n_{gr}}$, where $n_{gr} = 512$. For an implicit object representation $S(\mathbf{x}) = 0$, we note that a point $\mathbf{x}_0 \in \mathbb{R}^3$ is inside the object when $S(\mathbf{x}_0)$ is less than 0 and outside when $S(\mathbf{x}_0)$ is greater than 0. We use this fact to determine whether the gripper collides with the objects or tables or not: when the value

$$\min_j S(\mathbf{T}_{gr}^{-1}\mathbf{x}_{gr,j}), \tag{13}$$

where $\mathbf{T}_{gr}$ is the pose of the gripper, is less than 0, then the gripper collides with the object. Through this, collisions can be checked quickly and efficiently.

In real-world pushing manipulation experiments, the gripper is equipped with an Azure Kinect camera. To account for this, we also sample and use the camera point cloud to check the collision, as shown in the right of Figure 6. In this case, 1024 points are sampled (i.e., $n_{gr} = 1024$) on both the gripper and the camera.

Grasping criteria. Let the recognized shapes' implicit representations be $S_1(\mathbf{x}) = 0, S_2(\mathbf{x}) = 0, \cdots, S_N(\mathbf{x}) = 0$, and additionally, the table's implicit representation be $S_{N+1}(\mathbf{x}) = 0$ (the box-shaped table can also be represented by superquadric equation). Let $\mathbf{T}_{gr,1}, \cdots, \mathbf{T}_{gr,N_c} \in \mathrm{SE}(3)$ be the candidate grasp poses. Then the terminal cost is defined by:

$$q(\mathbf{s}_{T+1}) = 1 - \max_k \left[\mathbb{1}(\min_{i,j} S_i(\mathbf{T}_{gr,k}^{-1}\mathbf{x}_{gr,j}) > 0) \circ \mathbb{1}(\mathbf{T}_{gr,k} \text{ is kinematically feasible}) \right], \tag{14}$$

where $i = 1, \cdots, N+1$ is the object index, $j = 1, \cdots, n_{gr}$ is the gripper point cloud index, $k = 1, \cdots, N_c$ is the candidate grasp pose index, the indicator function $\mathbb{1}(\cdot)$ is N_c-dimensional vector, and $\circ$ is the element-wise multiplication. The terminal cost is 0 if at least one kinematically feasible and collision-free grasp pose exists and 1 otherwise. When the terminal cost achieves 0, grasping proceeds by selecting one of the grasp poses that satisfy both conditions, i.e., $\mathbf{T}_{gr,k}$ such that

$$\min_{i,j} S_i(\mathbf{T}_{gr,k}^{-1}\mathbf{x}_{gr,j}) > 0 \text{ and } \mathbf{T}_{gr,k} \text{ is kinematically feasible.} \tag{15}$$

D Experimental Details

D.1 Details for Pushing Manipulation Data Generation

This subsection includes further details when generating a pushing manipulation dataset for training pushing dynamics learning models.

Object configuration. The objects consist of cubes and cylinders with various shape parameters (i.e., width, height, depth for the cube, and radius, and height for the cylinder). The shape parameters are randomly generated, and 18 different objects are generated for each box and cylinder as shown in Figure 7. Then, the objects are divided into 9/9 known and unknown sets per shape class. The known objects are used for training the pushing dynamics models, and the unknown objects are used for the performance evaluation of the trained models. For data generation, these objects are dropped on the workspace of 0.512m $\times$ 0.512m $\times$ 0.192m.

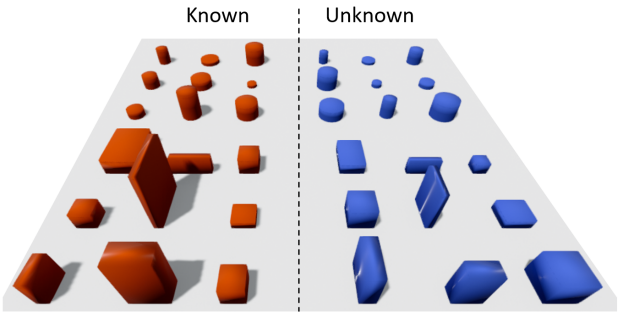

Figure 7: Known (red) and unknown (blue) object shapes used for data generation.

Action sampler. To sample an action $(\mathbf{p}, \mathbf{v}) \in \mathbb{R}^6$, we first randomly choose an object to push and randomly choose the pushing direction among pre-defined 8 angles divided equally between 0 to 2π towards the center of the object pose. The vector $\mathbf{v} \in \mathbb{R}^3$ is then defined from the pushing direction vector. To determine the pushing start point $\mathbf{p} \in \mathbb{R}^3$, the height of the pushing point is randomly chosen within the candidate heights which belong to 5 pre-defined heights divided equally by the workspace height (i.e., 0.192m) and less than the object's height. The starting point in the x-y plane is also chosen within the 4 candidates on the pushing line as shown in Figure 8. For each object, a maximum of 160 action candidates can be generated. For each action, the trajectory of the robot pushing motion is divided into 10 via points and the end effector of the robot reaches the via points sequentially and slowly, making the object as quasi-static as possible.

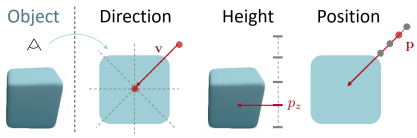

Figure 8: Pushing action sampling method for a chosen object.

D.2 Details for Pushing Dynamics Learning Experiments

Details for equivariance study. The models are trained with only one pushing manipulation data with a single object, which is a 3-tuple (past observation, pushing action, next observation). Then, the 9 pieces of test data are generated by applying random $\mathrm{SE}(2)$-transformation to the generated training data (i.e., the object and pose in the training data are subject to the same $\mathrm{SE}(2)$ transformation), and evaluate the trained model on this test dataset. This experiment is conducted on 10 training data with different objects and poses, so the total number of evaluations is 90 times.

Details for pushing dynamics learning. In this experiment, the models are trained with a relatively large dataset that consists of as follows: For each sequence, the objects are randomly chosen from the known objects, and randomly dropped on the workspace. The number of objects per sequence varies from 1 to 4. We sample and execute a maximum of 20 random pushing actions per sequence, and we stop the sequence when objects' positions are out of the workspace or objects fall down. We collect data until the total numbers of data tuples are 12000/1200/1200 for training/validation/test, respectively. To test the generalization performance for unknown objects, we additionally generate 300 data tuples per the number of objects from 1 to 4, where the objects are randomly chosen from the unknown objects. So, the unknown test dataset consists of 1200 data tuples. We evaluate the trained models on both the known and unknown test datasets. The segmentation, recognition, and motion prediction networks (SQPD-Net) take approximately 21.5 hours, 34 hours, and 16 hours to train on the RTX3080 Ti, respectively.

D.3 Details for Pushing Manipulation Experiments

Object configuration. For pushing manipulation experiments on simulation and real-world experiments, we use box- and cylinder-like objects inspired by the YCB dataset [49]. For each task, we use the object sets shown in Figure 9. In the moving and singulation tasks, a total of 10 experiments – 5 experiments for 2 object sets – are performed. In the grasping tasks, the "grasping large" and the "grasping clutter" tasks include 5 experiments for one object set. The objects have different poses for each experiment. Simulation manipulation experiments are conducted by making objects of the same size as these.

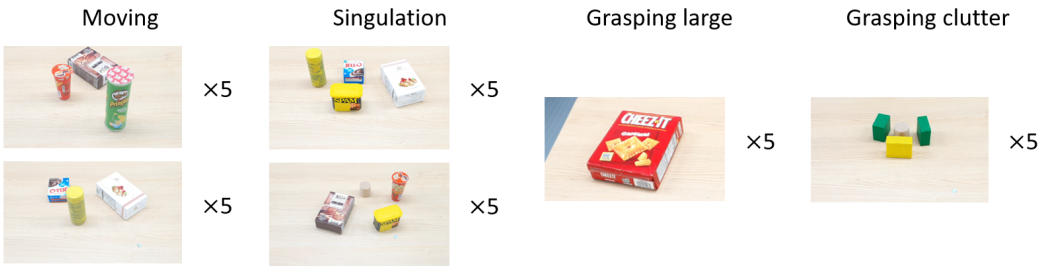

Figure 9: Object sets used in moving, singulation, and grasping tasks.

Action sampler. We use the same action sampler used in data generation as described in Section D.1 on the recognized object shapes. When sampling the pushing actions, we reject the actions that collide with the objects or the table are rejected using the gripper collision detection method introduced in Section C.3.

Details for pushing manipulation. We use the sampling-based MPCs [50]. At each timestep t, when the observation $\mathbf{o}_t$ (for our case, the partially observed point cloud $\mathcal{P}_t$) is obtained, the shape recognition is performed and obtain the shape parameters $\mathbf{q}_{t,i}$ and poses $\mathbf{T}_{t,i}$. We denote by $\mathbf{s}_t = \{(\mathbf{T}_{t,i}, \mathbf{q}_{t,i})\}_{i=1}^{N}$. From the recognized objects $\mathbf{s}_t$, we sample 100 action sequences; the time horizon of each sequence is one for moving and singulation tasks and three for grasping tasks. Then, using our trained SQPD-Net, we compute the next objects' poses (i.e., $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$) and accordingly compute cost functions using $\mathbf{s}_{t+1}$ for all sampled actions. Then we find an optimal action that best minimizes the cost function. The action sequences are resampled and the optimal action is chosen every timestep t. The success criterion for each task is as follows: (i) moving is success when the distance between the positions of the objects and the goal positions is less than 5cm on average, (ii) singulation is success when all distances between the objects are more than τ = 20cm, and (iii) grasping is success when at least one grasp pose is found. We note that the candidate grasp poses for the target object are resampled every timestep t. If the task trial does not succeed within 10 timesteps, the trial is considered as a failure.

E Additional Experimental Results

E.1 Additional Results for Equivariance Study

More examples of the results of the equivariance study in Section 4.1 are shown in Figure 10. The trend of the experimental results, including 2DFlow and SE3-Net, is similar to the experimental results in Section 4.1.

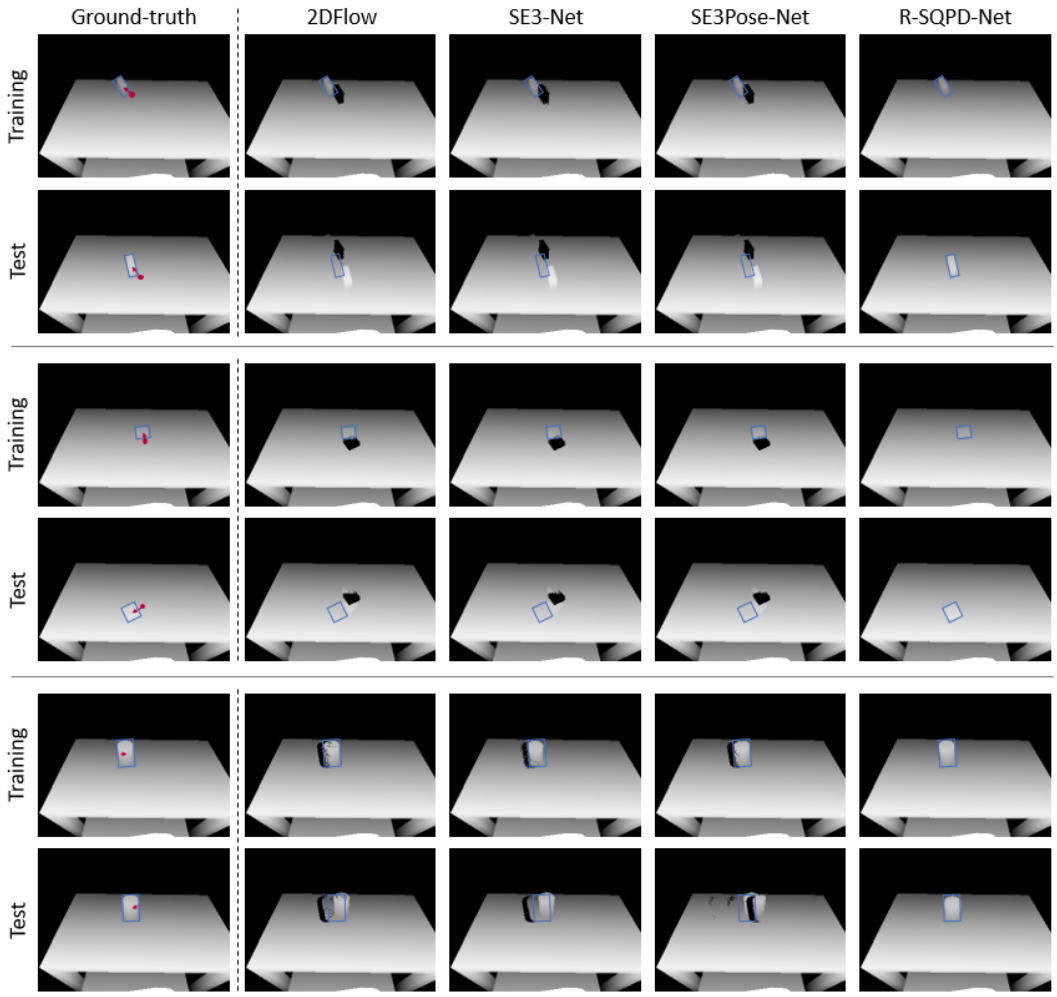

Figure 10: The representative three examples of the equivariance study experiments.

E.2 Additional Results for Pushing Dynamics Learning

More examples for the results of pushing dynamics learning in Section 4.2 are shown in Figure 11 and Figure 12. The trend of the experimental results is also similar to the experimental results in Section 4.2. The results of 2DFlow, SE3-Net, and SE3Pose-Net, which predict the flow of the point cloud, show that they have difficulties predicting the motions of the objects at all. 3DFlow and DSR-Net attempt to predict the motion more than the above methods, but the prediction accuracy is still not enough. Our SQPD-Net accurately predicts the motion of moving objects.

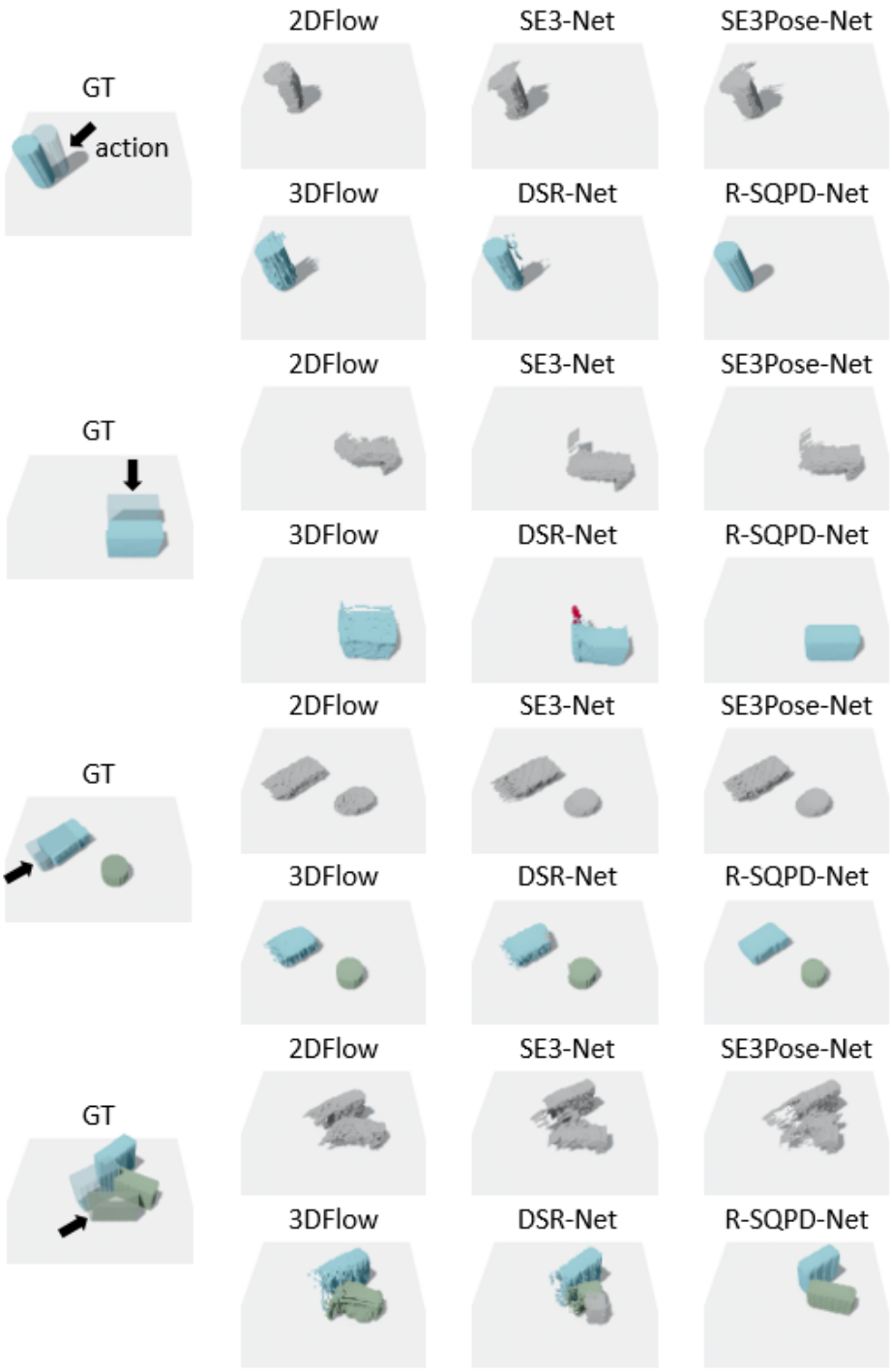

Figure 11: The representative examples of the pushing dynamics learning experiments for the number of objects 1 and 2.

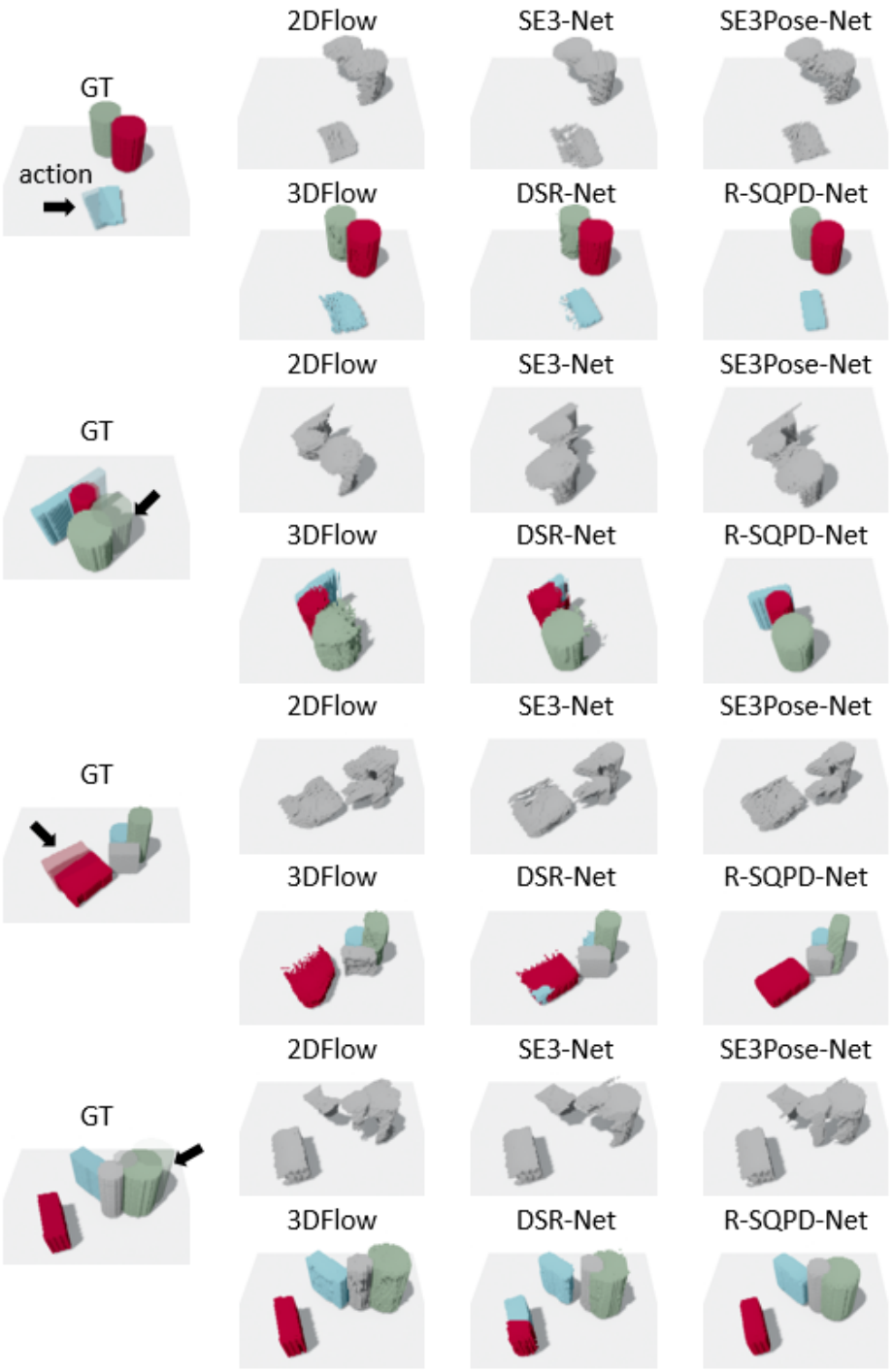

Figure 12: The representative examples of the pushing dynamics learning experiments for the number of objects 3 and 4.

E.3 Pushing Dynamics Learning on Real-world Pushing Data

In this experiment, we train our SQPD-Net on the real-world dataset and compare the performance with the physics-based simulator (PyBullet). We generate a real-world pushing manipulation dataset using four cube-shaped objects of different sizes shown in Figure 13. In this case, one scene contains only one of these four objects. For each scene, the object is placed in a fixed pose (rotated 22.5° degrees with respect to the robot's base frame), then we sample and execute one of the 20 different random pushing actions per object; so the total number of data tuples is 80. To annotate the pose of the object before and after the pushing action, we use the Iterative Closest Point (ICP) algorithm that matches the point cloud observation to the ground-truth object model. We then divide the four objects into three known objects and one unknown object; the known objects are used to train our R-SQPD-Net, and the unknown objects are used to evaluate the trained model. The number of cases of dividing objects is four, and the model is trained and evaluated in all these cases. We denote the model trained in box a, b, c and evaluated in box d as "$a, b, c \rightarrow d$". To get the predictions of PyBullet, we drop the superquadric object given by our trained recognition model into the simulator and let the robot simulator perform the same pushing action that is performed in the real-world.

Figure 13: Execution of pushing action.

Figure 14 shows the motion of the objects predicted by the PyBullet simulator and trained R-SQPD-Net. As shown, both PyBullet and R-SQPD-Net tend to predict the position and approximate direction of the objects after movement. However, R-SQPD-Net predicts the orientation of the predicted object much better. This is due to some differences between the dynamic natures of PyBullet and the real-world. Since our model is trained directly from data collected in the real-world, it can predict the dynamics of pushing objects well. In order to quantitatively verify this fact, translation and rotation errors are measured for the poses of the predicted objects as shown in Table 1. We have confirmed that R-SQPD-Net outperforms PyBullet overall, and especially, our model performs much better in terms of rotation errors. In conclusion, we verify that our model can be successfully trained on real-world data. Also, the simulator is somewhat accurate, but to perform more accurate pushing manipulations in the real-world, we should collect a dataset from the real-world and train the model on this dataset.

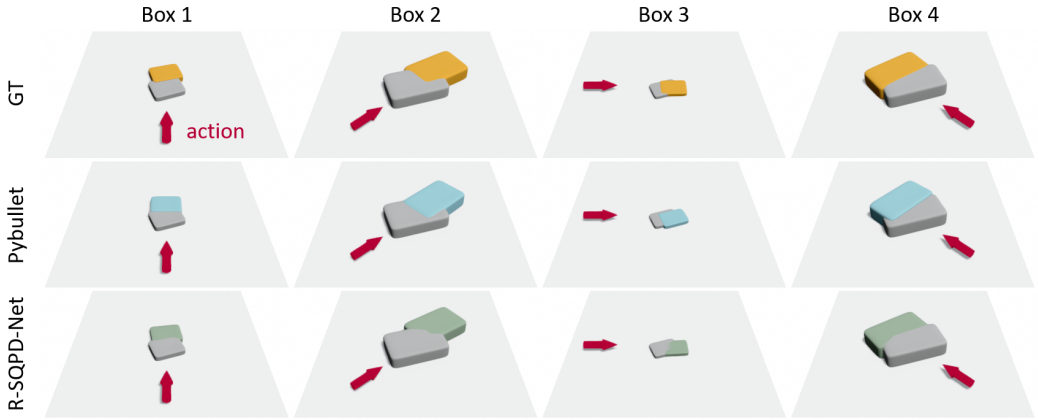

Figure 14: Real-world ground-truth pushing data (yellow) and pushing dynamics prediction results of PyBullet physics simulator (blue) and trained R-SQPD-Net (green). The initial pose of the object before being pushed is indicated in gray color.

Table 1: Translation and rotation errors computed with real-world data.

	PyBullet		R-SQPD-Net	
MODEL	translation (cm)	rotation (°)	translation (cm)	rotation (°)
2,3,4 → 1	1.025 ± 0.520	8.129 ± 7.206	**0.681 ± 0.349**	**4.304 ± 4.091**
1,3,4 → 2	1.007 ± 0.599	4.000 ± 3.658	**0.972 ± 0.580**	**3.581 ± 3.110**
1,2,4 → 3	0.963 ± 0.702	13.038 ± 7.357	**0.800 ± 0.347**	**4.197 ± 3.508**
1,2,3 → 4	0.847 ± 0.541	4.584 ± 3.308	**0.674 ± 0.512**	**2.995 ± 2.346**

E.4 Pushing Manipulation via Interaction

To verify that our model R-SQPD-Net works well for cases where multi-object interactions are essential to achieve the goal, we have conducted a new pushing manipulation task named **interactive moving**.

Interactive moving is similar to **moving** task in that the goal is to move some objects, but here it is a task that moves one target object to the desired pose. In this case, the robot *should not push the target object.* The current pose of the target object and the desired pose are given as $\mathbf{T_t}$ and $\mathbf{T}_d \in \mathrm{SE}(3)$ respectively, then we define a terminal cost function as

$$q(\mathbf{s}_{T+1}) = \|\mathbf{t}_{T+1} - \mathbf{t}_d\|_2^2, \tag{16}$$

where the notation $\mathbf{t}_{(\cdot)}$ denotes the translation vector of $\mathbf{T}_{(\cdot)}$. We set the desired positions $\mathbf{t}_d$ as $(0.3, 0.0, \mathbf{t}_{0,z})$; the task is described in the left of Figure 15. Since the robot cannot directly push the target object, it must move the target object to the desired position by pushing other objects. That is, this task essentially requires interaction between multi-objects. We sample 100 action sequences and the time horizon of each sequence is set to one.

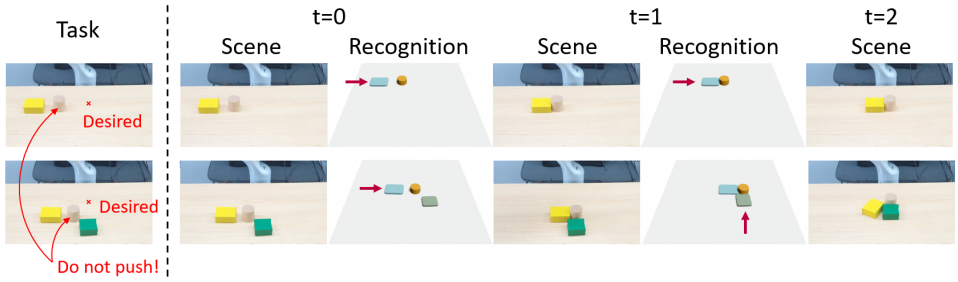

Figure 15: Real-world manipulation results using R-SQPD-Net for the interactive moving task (the target object is the cylinder surrounded by cubes). The red arrow at each recognition step means the optimal pushing action.

Figure 15 shows some manipulation results for interactive moving tasks. In the case of the first row, it succeeded in moving the target object to the desired position by pushing the yellow box. Even in the case of the slightly difficult case in the second row, our approach can find the series of actions that sequentially push the yellow and green boxes so that successfully perform the desired task. In conclusion, We verify that R-SQPD-Net can learn multi-object interaction well and that it can be used properly in manipulation tasks.

E.5 Supplementary Videos for Pushing Manipulation Experiments

Pushing manipulation videos can be found at https://www.youtube.com/watch?v=OLoAHhf7vk0.

E.6 Failure cases for Pushing Manipulation

There are some failure cases for pushing manipulation, and some representative examples of the failure cases are shown in Figure 16.

Figure 16: The representative examples of the failure cases for pushing manipulation.

The first case is the result of a failure to recognize the shape; this leads to inaccurate calculation of task objective function or unexpected collision with the objects (left of the Figure 16). Also, to prevent the object from falling down, we provide a constraint so that the center of the object is inside the workspace. The constraint sometimes did not work and the object falls down from the table since recognition and trained dynamics are not perfectly accurate (center of the Figure 16). The second case prevalent is the failure of sim-to-real transfer of the pushing dynamics model. Such cases occur when pushing the standing long cylinder, in detail, when the same pushing action is performed, the cylinder does not fall over in the simulation but sometimes falls over in the real-world experiment (right of the Figure 16). These unexpected object motions cause the performance drop of the pushing manipulation tasks.

References

[1] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[2] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg. Linear push policies to increase grasp access for robot bin picking. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1249–1256. IEEE, 2018.

[3] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong. Efficient learning of goal-oriented push-grasping synergy in clutter. *IEEE Robotics and Automation Letters*, 6(4):6337–6344, 2021.

[4] M. Kiatos and S. Malassiotis. Robust object grasping in clutter via singulation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1596–1600. IEEE, 2019.

[5] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119:119–134, 2019.

[6] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox. Nerp: Neural rearrangement planning for unknown objects. *arXiv preprint arXiv:2106.01352*, 2021.

[7] E. Huang, Z. Jia, and M. T. Mason. Large-scale multi-object rearrangement. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 211–218. IEEE, 2019.

[8] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork. Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9433–9440. IEEE, 2020.

[9] Y. Yang, H. Liang, and C. Choi. A deep learning approach to grasping the invisible. *IEEE Robotics and Automation Letters*, 5(2):2232–2239, 2020.

[10] M. Danielczuk, A. Angelova, V. Vanhoucke, and K. Goldberg. X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9577–9584. IEEE, 2020.

[11] A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE, 2017.

[12] A. Byravan, F. Leeb, F. Meier, and D. Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3339–3346. IEEE, 2018.

[13] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. Reasoning about physical interactions with object-oriented prediction and planning. International Conference on Learning Representations, 2019.

[14] Y. Ye, D. Gandhi, A. Gupta, and S. Tulsiani. Object-centric forward modeling for model predictive control. In *Conference on Robot Learning*, pages 100–109. PMLR, 2020.

[15] J. Wang, C. Hu, Y. Wang, and Y. Zhu. Dynamics learning with object-centric interaction networks for robot manipulation. *IEEE Access*, 9:68277–68288, 2021.

[16] B. Huang, S. D. Han, A. Boularias, and J. Yu. Dipn: Deep interaction prediction network with application to clutter removal. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4694–4701. IEEE, 2021.

[17] Z. Xu, Z. He, J. Wu, and S. Song. Learning 3d dynamic scene representations for robot manipulation. *arXiv preprint arXiv:2011.01968*, 2020.

[18] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. Shape completion enabled robotic grasping. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2442–2447. IEEE, 2017.

[19] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737. IEEE, 2018.

[20] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018.

[21] S. Liu, S. Saito, W. Chen, and H. Li. Learning to infer implicit surfaces without 3d supervision. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[23] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[24] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11516–11522. IEEE, 2020.

[25] T. E. Boult and A. D. Gross. Recovery of superquadrics from depth information. In *Proc. Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pages 128–137, 1987.

[26] A. Makhal, F. Thomas, and A. P. Gracia. Grasping unknown objects in clutter by superquadric representation. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 292–299. IEEE, 2018.

[27] G. Vezzani, U. Pattacini, and L. Natale. A grasping approach based on superquadric models. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1579–1586. IEEE, 2017.

[28] G. Vezzani, U. Pattacini, G. Pasquale, and L. Natale. Improving superquadric modeling and grasping with prior on object shapes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6875–6882. IEEE, 2018.

[29] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE transactions on pattern analysis and machine intelligence*, 12(2):131–147, 1990.

[30] S. Kim, T. Ahn, Y. Lee, J. Kim, M. Y. Wang, and F. C. Park. Dsqnet: A deformable model-based supervised learning algorithm for grasping unknown occluded objects. *IEEE Transactions on Automation Science and Engineering*, 2022.

[31] D. Paschalidou, A. O. Ulusoy, and A. Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10344–10353, 2019.

[32] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[33] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[35] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[36] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[37] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[38] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018.

[39] T. S. Cohen, M. Geiger, and M. Weiler. A general theory of equivariant cnns on homogeneous spaces. *Advances in neural information processing systems*, 32, 2019.

[40] E. van der Pol, D. Worrall, H. van Hoof, F. Oliehoek, and M. Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4199–4210, 2020.

[41] D. Wang, R. Walters, X. Zhu, and R. Platt. Equivariant q learning in spatial action spaces. In *Conference on Robot Learning*, pages 1713–1723. PMLR, 2022.

[42] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400. IEEE, 2022.

[43] H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[44] Y. Lee, J. Baek, Y. M. Kim, and F. C. Park. Imat: The iterative medial axis transform. In *Computer Graphics Forum*, volume 40, pages 162–181. Wiley Online Library, 2021.

[45] Y. Lee, S. Kim, J. Choi, and F. Park. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*, pages 12378–12402. PMLR, 2022.

[46] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[47] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[48] A. D. Gross and T. E. Boult. Error of fit measures for recovering parametric solids. In *ICCV*, 1988.

[49] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[50] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.